ORNL Accelerator Control Systems with uTCA and Buildroot

Matt Waddel

EPICS Collaboration Meeting - September 2024

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Reasons for Switching to Buildroot

- Deployed operating system inconsistency were becoming a problem
  - Red Hat, Debian and version differences
  - Everyone was choosing their favorite
- Hardware differences between installed systems
  - Mostly a memory size variance on Vadatech CPUs
- Hardware failures took many hours to recover
- Recovering failed systems was a manual process that was prone to mistakes

ORNL uTCA System Requirements

- Must be able to replace a failed CPU in less than 10 minutes
- Automatically start the IOC on system boot
- Minimize boot image size to facilitate network booting
  - Ideally less than 50MBytes
- Root image loaded via the network
  - NFS mount EPICS and IOC executables
- Toolchain, libraries and supporting binaries are shared via an NFS
- Systems builds in remote environment; development or production

Buildroot Installation Steps

- Buildroot Linux installation overview
  - https://buildroot.org/downloads/manual/manual.html
  - Install required host tools, many of these tools are already installed
  - IT policy requires package installation from ORNL sponsored repository

Red Hat

Required Packages

```
sudo yum install sed make binutils diffutils gcc-c++ bash patch gzip bzip2 perl tar cpio unzip \
                 rsync file bc findutils wget ncurses-devel qt5-* perl-ExtUtils-MakeMaker openssl-devel
```

Optional Packages

```
sudo yum install git mercurial rsync subversion asciidoc dblatex graphviz pkg-config
```

Download LTS Buildroot

```
mkdir –p /ics/embedded/ && cd /ics/embedded/
curl https://code-int.ornl.gov/ics/embedded/buildroot-repo/-/raw/main/buildroot-2023.02.6.tar.gz | tar zxv
```

Link the installed directory

```
ln -s buildroot-2023.02.6 buildroot
unset LD_LIBRARY_PATH
```

Debian/Ubuntu

Required Packages

```
sudo apt install sed make binutils build-essential diffutils gcc g++ bash patch gzip bzip2 perl tar cpio unzip \
                 rsync file bc findutils wget libncurses-dev libelf-dev libssl-dev qt5-*
```

Optional Packages

```
sudo apt install python cvs git mercurial rsync subversion asciidoc w3m dblatex graphviz
```

Download LTS Buildroot

```
mkdir –p /ics/embedded/ && cd /ics/embedded/
curl https://code-int.ornl.gov/ics/embedded/buildroot-repo/-/raw/main/buildroot-2023.02.6.tar.gz | tar zxv
```

Link the installed directory

```
ln -s buildroot-2023.02.6 buildroot
unset LD_LIBRARY_PATH
```

Buildroot ORNL Customizations

- ORNL customizations include custom packages, kernel configuration, packages, system setup, etc.

- The configuration is tracked in a repository that is checked-out and applied over the default Buildroot distribution

```
cd /ics/embedded
git clone https://code-int.ornl.gov/ics/embedded/buildroot-cfg  ## This step also creates the buildroot-cfg directory
cd /ics/embedded/buildroot
make ics-dev_defconfig
```

- SNS changes are now applied. Use this step if further customizations are needed or check existing configuration

```
cd /ics/embedded/buildroot && make menuconfig
```

- Changes are not automatically synchronized to the SNS configuration folder
  Transfer additional Buildroot config changes with:

```
make savedefconfig BR2_DEFCONFIG=/ics/embedded/buildroot-cfg/configs/ics-dev_defconfig
```

- Changes to the Linux kernel

```
make linux-menuconfig
```

- Make sure to copy Linux .config file manually to the kernel configuration

```
cp output/build/linux-xxx/.config /ics/embedded/buildroot-cfg/board/ics-dev/kernel.config
```

- Build a new image

```
cd /ics/embedded/buildroot/ && make -j8
```

Testing Buildroot Images Locally

- Using QEMU to verify built images

```
sudo apt install qemu qemu-system
mkdir /tmp/test
cd /tmp/test
cp /ics/embedded/buildroot/output/images/* .

qemu-system-x86_64 -M pc -kernel ./bzImage -drive file=rootfs.tar,format=raw -append "rootwait root=/dev/vda console=ttyS0" -net nic,model=virtio -net user -nographic -m 512  # --enable-kvm
<cntrl>A x
```

(Partial Boot Messages)

```
[    0.000000] Linux version 6.1.57 (lxuser@dev-opi2) (x86_64-buildroot-linux-gnu-gcc.br_real (Buildroot 2023.02.6) 11.4.0, GNU ld (GNU Binutils) 2.38) #91 SMP PREEMPT_DYNAMIC Mon Jul  1 13:26:34 EDT 2024
[    0.000000] Command line: rootwait root=/dev/vda console=tty1 console=ttyS0
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] BIOS-provided physical RAM map:
...
[    1.149146] Run /init as init process
mount: mounting 192.168.201.175:/data/ics on /ics failed: Network is unreachable
Saving 256 bits of non-creditable seed for next boot
Starting syslogd: OK
Starting klogd: OK
[    2.206606] udevd[166]: starting eudev-3.2.11
done
Missing IP address, aborting
Starting network: OK
Starting chrony: OK
Setting initial time: 200 OK
Starting dropbear sshd: OK
mount: mounting 192.168.201.175:/data/ics on /ics failed: Network is unreachable
Starting cron ... done.

Welcome to SNS MicroTCA Linux
utca login:
```

Adding EPICS Linux Device Drivers

- In the buildroot-cfg directory that was created in a previous step, add the driver source code

```
cd /ics/embedded/buildroot-cfg && mkdir utcaInjKickDriver && cd utcaInjKickDriver
```

- 2 files are needed in this directory: Config.in and utcaInjKickDriver.mk

```
cat Config.in
  config BR2_PACKAGE_UTCAINJKICKDRIVER
    bool "utcaInjKickDriver"
    help
        Linux kernel driver for Injection Kicker Monitor Kernel Driver

cat utcaInjKickDriver.mk
  ##################################################################################
  #
  # injKickDriver
  #
  ##################################################################################
  UTCAINJKICKDRIVER_VERSION = R1-0-9
  UTCAINJKICKDRIVER_SITE_METHOD=git
  UTCAINJKICKDRIVER_SITE = https://code-int.ornl.gov/ics/embedded/drivers/utcaInjKickDriver
  UTCAINJKICKDRIVER_LICENSE = GPL
  UTCAINJKICKDRIVER_LICENSE_FILES = LICENSE
  UTCAINJKICKDRIVER_MODULE_SUBDIRS = driver

  $(eval $(kernel-module))
  $(eval $(generic-package))
```

- This Linux device driver will be checked out from the git repository, built and installed to /lib/modules/<kernel version>/<directory>/

- Custom build and installation commands can be added, but it's usually best to let Buildroot "make" system do its job

Editing and Rebuilding a Linux Device Driver

- Making temporary changes in the build directory

```
cd /ics/embedded/buildroot/output/build/utcaInjKickDriver-R1-0-9/driver
cd ../../../..
make utcaInjKickDriver-rebuild
cd /ics/embedded/buildroot/output/build/utcaInjKickDriver-R1-0-9/driver
cp injKickWF0050.ko /ics/tmp/ && cd /ics/tmp/
rmmod injKickWF0050 && insmod ./injKickWF0050.ko
```

- The actual git repository checkout can be found in this directory

```
cd /ics/embedded/buildroot/download/utcaInjKickDriver/git/driver/
git commit –a –m "New feature"
git tag R1-0-10
git push && git push --tags
```

- To force a new checkout and rebuild of a driver (any changes will be lost)

```
cd /ics/embedded/buildroot/
rm –rf /ics/embedded/buildroot/output/build/utcaInjKickDriver-R1-0-9/
make utcaInjKickDriver <or>
make utcaInjKickDriver-rebuild
```

Adding a Customized Buildroot Package

- Very silimar to the process for adding Linux Device Driver

```
cd /ics/embedded/buildroot-cfg/package/
mkdir procServ && cd procServ
```

- Add a Config.in entry

```
config BR2_PACKAGE_PROCSERV
    bool "procServ"
    depends on BR2_USE_MMU # fork()
    help
      A wrapper to start arbitrary interactive commands in the background,
       with telnet access to stdin/stdout.
       https://github.com/ralphlange/procServ
```

- Add a procServ.mk file

```
################################################################################
#
# procServ
#
################################################################################
PROCSERV_VERSION = 2.8.0
PROCSERV_SITE = https://github.com/ralphlange/procServ/releases/download/v$(PROCSERV_VERSION)
PROCSERV_SOURCE = procServ-$(PROCSERV_VERSION).tar.gz
PROCSERV_LICENSE = GPL
PROCSERV_LICENSE_FILES = COPYING
$(eval $(autotools-package))
```

- Select the procServ option in the menuConfig

```
...
BR2_PACKAGE_PROCSERV=y
...
```

PXE booting (pxelinux and iPXE)

- Initial network boot development with PXE booting
- Older PXE boot process didn't work with newer BIOS
- iPXE works well with new BIOS
  - loads faster
  - Simplified setup (doesn't need MAC address addition)
- Tried to make boot system similar to a VME boot process
- Uses a temporary IP Address to download image and start booting
- Utilizes consistency in Vadatech MAC Addresses (00:13:3a:XX:XX:XX)

PXE Linux setup

- Setup a PXE boot-image directory

```
mkdir –p /ics/boot/utca/pxelinux.cfg/ && cd /ics/boot/utca/pxelinux.cfg/
vi 01-00-13-3a-xx-xx-xx ## Red highlight is MAC address of board
-----
default utca
label utca
  kernel images/bzImage
  append net.ifnames=0 utca.net.dev=eth2 utca.net.ip=192.168.200.220 utca.net.mask=255.255.254.0 utca.net.dns=192.168.200.100 utca.net.gw=192.168.200.1
```

- Create a directory for the boot images

```
mkdir –p /ics/boot/utca/images && cd /ics/boot/utca/images
cp /ics/embedded/buildroot/output/image/bzImage .
```

- Setup DHCP to serve IP Address and Image Directory

```
cd /etc/dhcp/ && vi dhcpd.conf
class "MicroTCA" {
    match if substring (hardware,1,3) = 00:13:3a
}
...
pool {
            allow members of "MicroTCA";
            range 192.168.201.45 192.168.201.49;
            default-lease-time 30;
            max-lease-time 60;
            server-name "192.168.201.213";
            next-server 192.168.201.213;
            if option arch = 00:07 or option arch = 00:09 {
              filename "/ics/boot/utca/bootx64.efi";   ### iPXE setup
            }
            else {
              filename "/ics/boot/utca/pxelinux.0";
            }
        }
```

- Restart the DHCP server process

```
/etc/init.d/dhcpd restart
```

iPXE Boot setup

- Download source from https://ipxe.org/download - git clone https://github.com/ipxe/ipxe.git

  ```
  cd ipxe/src
  make
  ```

- Customizing the iPXE boot script with a generic script

  ```
  cd ~/controls_integration/ipxe/src
  vi mac-boot-generic
  ---
     #!ipxe
     echo "Starting iPXE ------- Setup DHCP boot"
     dhcp net0
     echo "CPU MAC Address detected: "${mac}
     set BootServer tftp://192.168.201.175
     echo "BootServer selected: "${BootServer}
     echo "Load board specific script: "${BootServer}/utca/ipxelinux.cfg/${mac}
     imgload ${BootServer}/utca/ipxelinux.cfg/${mac}
     sleep 1
     boot
  ```

- Build and Deploy Instructions

  ```
  make clean; make bin-x86_64-efi/ipxe.efi EMBED=mac-boot-generic
  scp bin-x86_64-efi/ipxe.efi root@192.168.201.142:/ics/boot/utca/bootx64.efi
  ```

iPXE Boot setup (cont)

- Setup a custom boot script directory similar to the PXE boot

```
mkdir /ics/boot/utca/ipxe.cfg/
vi 00:13:3a:xx:xx:xx  # This does not need the 01- prefix or "-" in place of ":"
   #!ipxe
   echo "Starting ipxe boot system"
   show mac
   set BootServer tftp://192.168.201.175
   echo ${BootServer}
   kernel ${BootServer}/utca/images/ring-inj-wfgen memmap=600M@4G net.ifnames=0 utca.net.dev=eth2 utca.net.ip=192.168.201.142 \
            utca.net.mask=255.255.254.0 utca.net.dns=192.168.200.100 utca.net.gw=192.168.200.1
   boot
```

- Boot directive found in dhcpd.conf (allows old and new PXE booting)

```
if option arch = 00:07 or option arch = 00:09 {
                     filename "/ics/boot/utca/bootx64.efi";   ### iPXE setup
                  }
```

Passing Parameters to Buildroot System

- PXELinux Boot – Kernel Command Line

```
...
append memmap=600M@4G net.ifnames=0 utca.net.dev=eth2 utca.net.ip=192.168.201.13 utca.net.mask=255.255.254.0 utca.net.dns=192.168.200.100 utca.net.gw=192.168.200.1
```

- iPXE – Kernel Command Line

```
...
kernel ${BootServer}/utca/images/ring-inj-wfgen memmap=600M@4G net.ifnames=0 utca.net.dev=eth2 utca.net.ip=192.168.201.142 \
utca.net.mask=255.255.254.0 utca.net.dns=192.168.200.100 utca.net.gw=192.168.200.1
```

- Startup Script in Builtroot Image - /etc/init.d/S40netconf

```
  A simple script to configure network settings based on Linux kernel command
  line parameters. It works for a single network interface only.
  Script will write /etc/network/interfaces and /etc/resolv.conf files
  and expect the system to actually set-up the network by established ifup/ifdown
  When called at boot time, it must be called before /etc/init.d/S40network.
  Recognized parameters are:
    - utca.net.dev - Network interface to use, defaults to eth0
    - utca.net.ip - Device IP address, ie. 192.168.201.214
    - utca.net.mask - Network mask, ie. 255.255.254.0
    - utca.net.gw - Default gateway, ie. 192.168.201.1
    - utca.net.dns - Comma separated list of DNS nameservers, ie. 192.168.201.174,192.168.201.175
```

Passing Parameters to Buildroot System (cont)

```
#!/bin/bash

DEV="eth0"
IP=""
MASK=""
GW=""
DNS=""
args=`cat /proc/cmdline`
...
case "${1}" in
  start)
    for arg in $args; do
      if [[ $arg == utca.net.* ]]; then
        param=`echo $arg | sed 's/.*\.\(.*\)=.*/\1/' | tr 'a-z' 'A-Z'`
        value=`echo $arg | sed 's/.*=\(.*\)$/\1/'`
        eval "$param=\$value"
      fi
    done
    # Perform some basic sanity checks
    [ -z $IP ]   && echo "Missing IP address, aborting" && exit 1
    [ -z $MASK ] && echo "Missing MASK address, aborting" && exit 1
    # Finally write network settings to the system configuration
    echo "auto $DEV"                >> /etc/network/interfaces
    echo "iface $DEV inet static"  >> /etc/network/interfaces
    echo "address $IP"              >> /etc/network/interfaces
    echo "netmask $MASK"            >> /etc/network/interfaces
    if [ ! -z $GW ]; then
      echo "gateway $GW"            >> /etc/network/interfaces
    fi
    # Configure the DNS
    if [ ! -z $DNS ]; then
      echo "search ics.sns.gov" > /etc/resolv.conf
      for dns in `echo $DNS | tr ',' ' '`; do
        echo "nameserver $dns" >> /etc/resolv.conf
      done
    fi
    ;;
...
```

Automated IOC startup

- Works with single purpose systems, ie. only one IOC runs
- Make sure hostname matches the IOC name
- Remember to NFS mount the main directory
- Make sure the NFS IOC directory matches the hostname and the IOC name:
  /ics/iocs/utca/ring-ext-ioc-wfmon/...
  /ics/var/ring-ext-ioc-wfmon/log...
  hostname: ring-ext-ioc-wfmon.ics.sns.gov
- /etc/init.d/S90ioc parses and starts the IOC

S90ioc

```
#!/bin/sh
IOC_NAME=`hostname -s`
IOC_DIR=/ics/iocs/utca/$IOC_NAME
IOC_BOOT_DIR=$IOC_DIR/iocBoot
IOC_EXEC=$IOC_BOOT_DIR/st.cmd
PID_FILE=/ics/var/$IOC_NAME/procServ.pid
LOG_FILE=/ics/var/$IOC_NAME/ioc.log
CORE_FILE=/ics/var/$IOC_NAME/coredump.%p
TZ=`cat /etc/timezone`
[ -d $IOC_DIR ] || exit 1

function configLogRotate() {
        cat << EOF > /etc/logrotate.d/ioc.conf
${LOG_FILE} {
    rotate 7
    ...
    endscript
    su lxuser epics
}
EOF
}
```

(Continued on next slide)

S90ioc (cont)

```
case "${1}" in
    start)
        # Raise limits for the process priorities and memory limits
        # Unfortunately BusyBox doesn't provide a prlimit command
        # for setting process limits, instead we set it for every
        # process on the system but IOC should really be the only one
        # running anyway.
        ulimit -r 99 -e 99 -l 10000000000 || exit 1

        # Enable core dumps when IOC crashes.
        # Must use `su` as `sudo` doesn't inherit the settings created
        # here. Because we run the use as lxuser, we must change the
        # `suid_dumpable` setting. lxuser must have write permission
        # to the core dump location.
        ulimit -c unlimited
        echo $CORE_FILE > /proc/sys/kernel/core_pattern
        echo 1          > /proc/sys/fs/suid_dumpable

        su lxuser -c "/usr/bin/procServ -c $IOC_BOOT_DIR -p $PID_FILE -L $LOG_FILE --logstamp 5000 $IOC_EXEC" || exit 1
        configLogRotate `cat $PID_FILE`
        ;;
    stop)
        kill `cat $PID_FILE` || exit 1
        ;;
    restart)
        $0 stop
        $0 start
        ;;
    console)
        telnet 127.0.0.1 5000
        ;;
    *)
        echo "Usage: $0 {start|stop|restart|console}"
        exit 1
        ;;
esac
```

Conclusion

- Once the basic Buildroot setup was complete adoption is happening quickly
- Boot time and startup complexity has been reduced
- Haven't had to test the 10-minute replacement rule, but I'm confident it is possible
- Buildroot imposed restrictions have improved reliability and maintainability

- Acknowledgment: Klemen Vodopivec(main architect for ORNL Buildroot system)